DESCRIPTION :
The ADC is configured to operate according to the configuration pattern. The configuration pattern is determined by the WARP signal which when set, is optimized for fast readout and when not set, is configured for slow readout and optimized for low noise.
This module clocks the 18 bit serial data out from ADC into an 18 bit shift register after or during each conversion cycle. Which clock (sysclk or ADC clock) is specified by the EXT configuration bit of each channel. This bit is currently locked out by in the schematic. Only internal clock mode is used. Which read mode (read after convert or read during convert) is controlled by the WARP bit. Hence, when WARP mode is selected, the internal ADC clock is used to clock data out during the next conversion time.
If WARP is not selected, the adc data is read out after the current conversion.
IMPLEMENTATION NOTES :

THERE ARE FOUR BITS USED FOR CHANNEL IDENTITY IN THE CONFIGURATION REGISTER. THESE BITS ARE LOADED INTO THE PHYSICAL ADC CHANNEL CONFIGURATION REGISTER AND DETERMINE WHICH CDS COMMAND CHANNEL THIS PHYSICAL CHANNEL LISTENS TO. THE CDS COMMAND DATA IS WRITTEN TO THE 38 BIT BUS CONNECTING THE CPLDS TO THE ACQUISTION FPGA WITH BIT FLAGS SET TO DETERMINE WHICH CDS CHANNEL THESE COMMANDS ARE DIRECTED TO. IF THE CHANNEL IDENT WRITEN TO THE ADC CONFIG REGISTER MATCHES THE BIT POSITION IN THE CDS COMMAND WORD THEN THAT COMMAND IS CARRIED OUT BY THE PHYSICAL ACQUISITION CHANNEL.

CONFIGURATION DATA BIT SIGNIFICANCE IS:

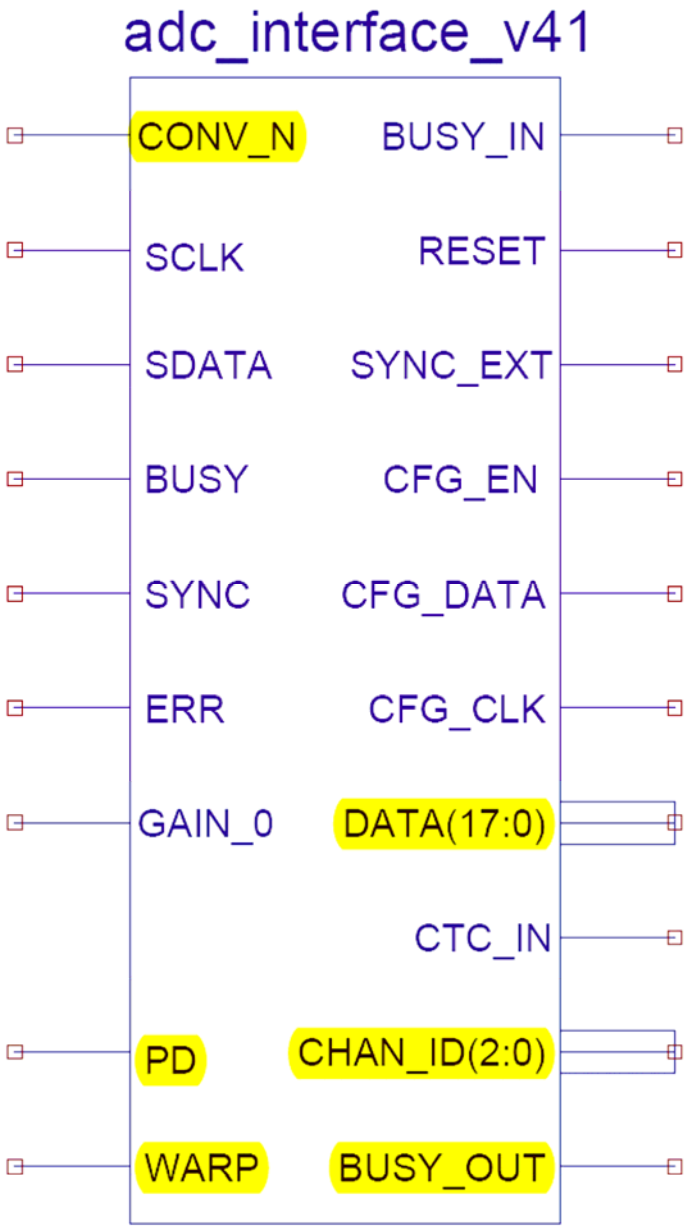
BIT
0 = POWER DOWN ADC AND LOCK CDS
1 = USE ADC WARP MODE FOR READOUT
2 = GAIN BIT 0 FOR CDS DC GAIN SETTING
3 = NOT USED IN THIS VERSION
4 = CHANNEL IDENT BIT 0
5 = CHANNEL IDENT BIT 1

BIT
6 = CHANNEL IDENT BIT 2
7 = NOT USED EXCEPT FOR READ BACK
8 = NOT USED EXCEPT FOR READ BACK
9 = READ BACK ADC STATUS AND CONFIG REG
10 = READ SDATA TEST STREAM INSTEAD OF ADC DATA

CONFIGURATION REGISTER CONTENTS AND THE RAW ADC STATUS SIGNALS CAN BE READ BACK BY SETTING THE CONFIGURATION BIT 9. THE RETURNED DATA REFLECTS THE FOLLOWING INFORMATION:

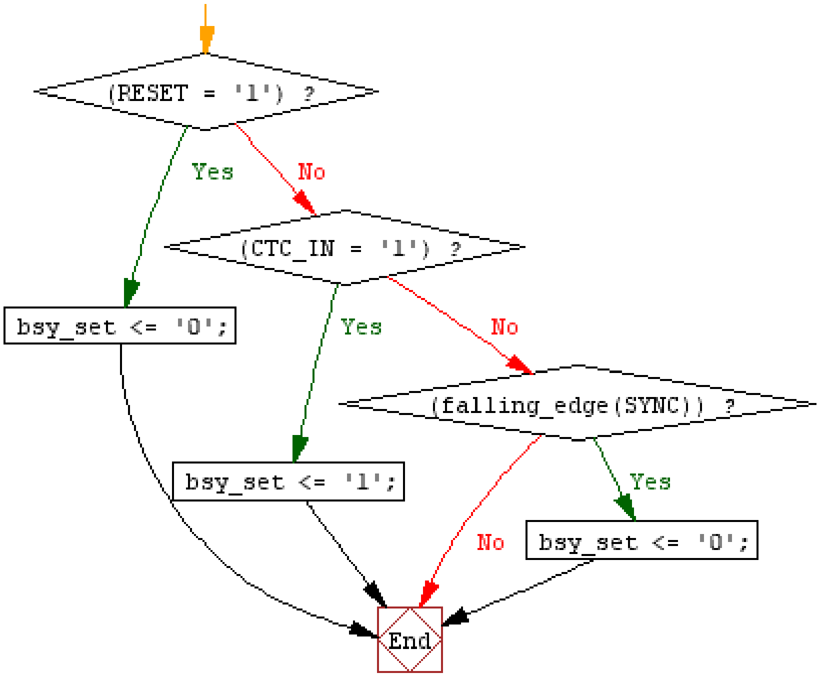
BIT
0 = SYNC SIGNAL FROM ADC
1 = BUSY SIGNAL FROM ADC
2 = SERIAL DATA STATE FROM ADC
3 = ERROR BIT FROM ADC
4 = GAIN BIT 0 OF CDS
5 = SPARE BIT IN ADC CONFIG REG
6 = ALWAYS ZERO
7 = ALWAYS ZERO
8 = CHANNEL IDENTITY BIT 0 (LSB)

BIT
9 = CHANNEL IDENTITY BIT 1
10 = CHANNEL IDENTITY BIT 2 (MSB)
11 = ALWAYS ZERO
12 = ALWAYS ZERO
13 = ALWAYS ZERO
14 = ADC_MODE SIGNAL
15 = POWER DOWN BIT ECHO
16 = ALWAYS ZERO
17 = ALWAYS ZERO

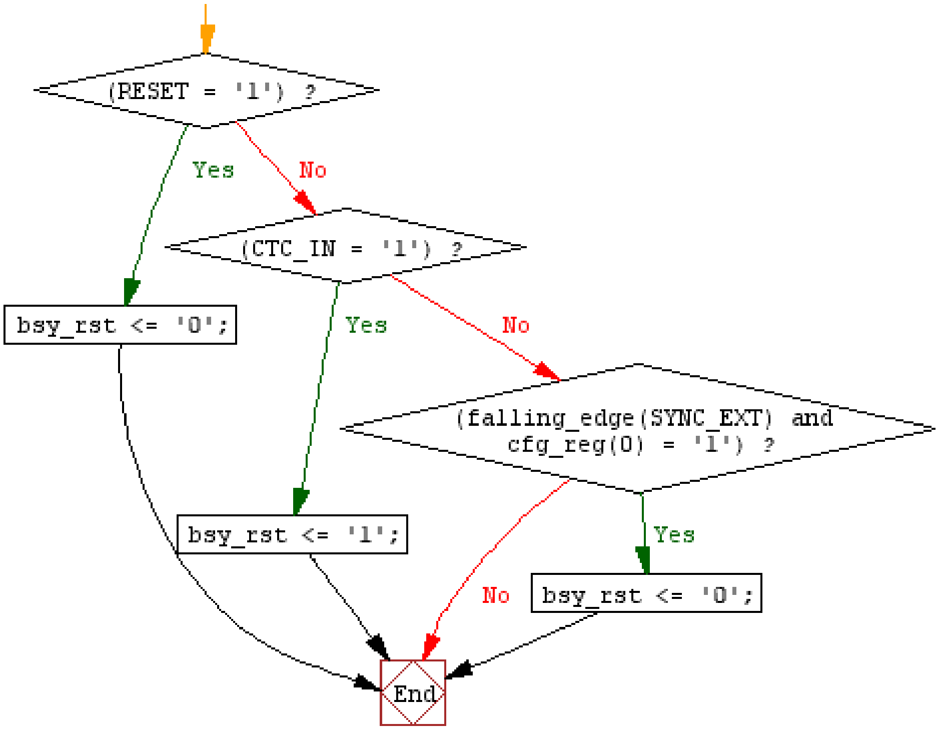


```
BUSY     : in  std_logic;
BUSY_IN  : in  std_logic;
BUSY_OUT : out std_logic;
CFG_CLK  : in  std_logic;
CFG_DATA : in  std_logic;
CFG_EN   : in  std_logic;
CHAN_ID  : out std_logic_vector(2 downto 0)
CONV_N   : out std_logic;
CTC_IN   : in  std_logic;
DATA     : out std_logic_vector (17 downto 0);
ERR      : in  std_logic;
GAIN_0   : out std_logic;
PD       : out std_logic;
RESET    : in  std_logic;
SCLK     : in  std_logic;
SDATA    : in  std_logic;
SYNC     : in  std_logic;
SYNC_EXT : in  std_logic;
WARP     : out std_logic;
```

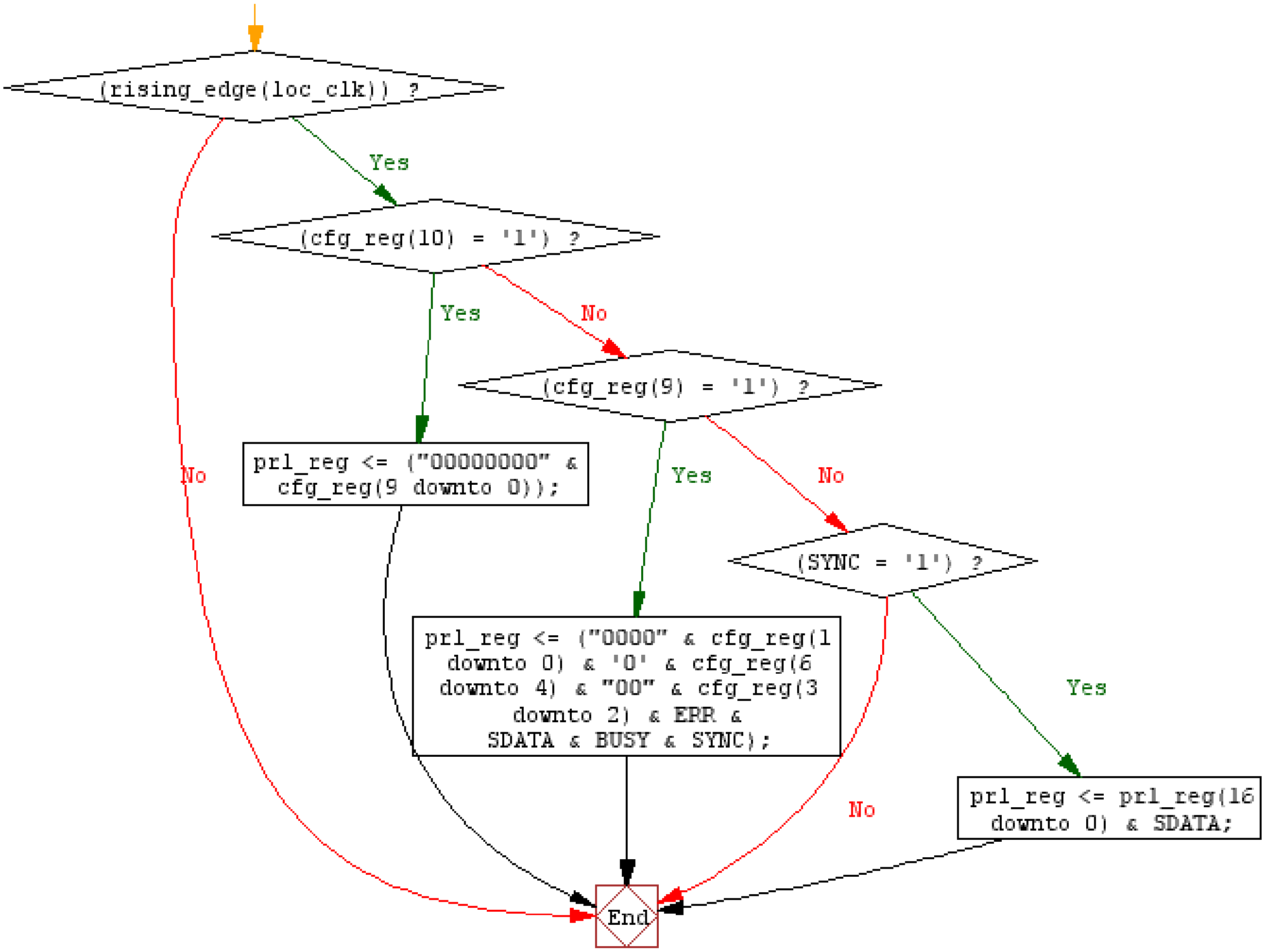
```
    BusySetProc: process(CTC_IN, RESET, SYNC)
```



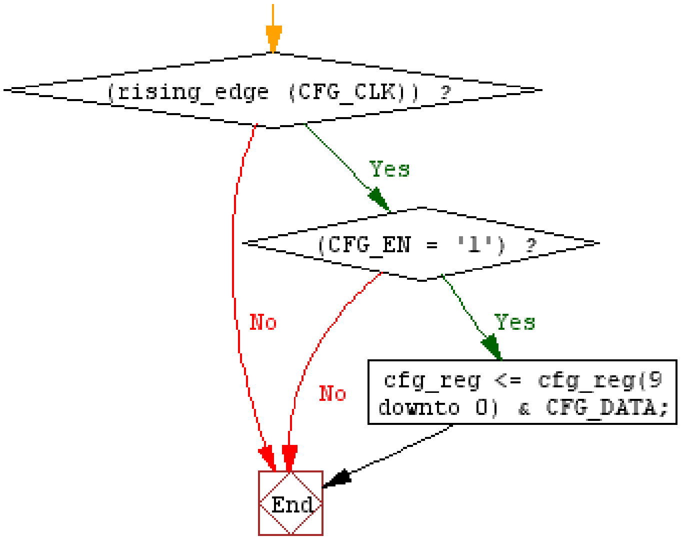
```
    BusyRstProc: process (CTC_IN, RESET, SYNC_EXT, cfg_reg(0))
```



AdcDataRegProc: process (loc_clk, cfg_reg, ERR, SDATA, BUSY, SYNC)


(rising_edge(loc_clk)) ?
Yes
No
(cfg_reg(10) = '1') ?
Yes
No
(cfg_reg(9) = '1') ?
prl_reg <= ("00000000" & cfg_reg(9 downto 0));
Yes
No
(SYNC = '1') ?
prl_reg <= ("0000" & cfg_reg(1 downto 0) & '0' & cfg_reg(6 downto 4) & "00" & cfg_reg(3 downto 2) & ERR & SDATA & BUSY & SYNC);
Yes
No
prl_reg <= prl_reg(16 downto 0) & SDATA;
End

AdcConfSerProc: process(CFG_CLK)


(rising_edge (CFG_CLK)) ?
Yes
(CFG_EN = '1') ?
No
Yes
No
cfg_reg <= cfg_reg(9 downto 0) & CFG_DATA;
End